A Framework for Control and Observation in Distributed Environments

Warren Smith

Computer Sciences Corporation
NASA Ames Research Center
Mail Stop T27A-2
Moffett Field, CA 94035
wwsmith@nas.nasa.gov

NAS Technical Report Number: NAS-01-006

June 2001

Abstract

As organizations begin to deploy large computational grids, it has become apparent that systems for observation and control of the resources, services, and applications that make up such grids are needed. Administrators must observe the operation of resources and services to ensure that they are operating correctly and they must control the resources and services to ensure that their operation meets the needs of users. Further, users need to observe the performance of their applications so that this performance can be improved and control how their applications execute in a dynamic grid environment. In this paper we describe our software framework for control and observation of resources, services, and applications that supports such uses and we provide examples of how our framework can be used.

1. Introduction

A recent trend in government and academic research is the development and deployment of computational grids [12, 17]. Computational grids are large-scale distributed systems that typically consist of high-performance compute, storage, and networking resources. Examples of such computational grids are the DOE Science Grid [3], the NASA Information Power Grid [8, 22], and the NSF Partnerships for Advanced Computing Infrastructure [9, 10]. Most of the work to deploy these grids is in developing the software services to allow users to execute applications on large and diverse sets of distributed resources. These services include security, execution of remote applications, managing remote data, access to information about resources and services, and so on. There are several toolkits that provide these services, such as Globus [5, 16], Legion [7, 19], and Condor [2, 23].

NASA is building a computational grid called the Information Power Grid (IPG) that is based upon the Globus toolkit. The IPG currently consists of resources and users at four NASA centers and our attempt to deploy a production grid of this size has highlighted the need for systems to observe and control the resources, services, and applications that make up such grids. We have found it difficult to ensure that the many resources in the IPG and the grid services executing on those resources are performing correctly. We have also found it cumbersome to perform

administrative tasks such as adding grid users to our resources. These observations have led to our development of a software framework to address these needs.

This paper provides an overview of our framework that provides a secure, scalable, and extensible framework for making observations on remote computer systems, transmitting this observational data to where it is needed, performing actions on remote computer systems, and analyzing observational data to determine what actions should be taken. The next section will provide an overview of our system. Section 3 provides two examples of how our framework can be used. Section 4 provides a summary and describes our planned future work.

2. CODE Framework

We have named our framework for Control and Observation in Distributed Environments CODE, for obvious reasons. This section describes the architecture of our system and its implementation.

2.1. Architecture

We call the software that we are developing a framework because it contains the core code that is necessary for performing monitoring and management. Users only need to add components to this framework and start the framework running. For example, if a user wants to create a host monitor, she would create components to monitor processes, files, network communications, and so on. The user would then add these components to the framework and tell the framework to begin monitoring the host. This same process is used for adding components to perform management actions. In fact, the typical process will be easier because CODE provides a set of commonly used components for observing various properties and performing various actions and all a user will have to do is select which of these components to use.

The CODE architecture is shown in Figure 1. The components that are shown with a solid outline are those that are supplied by our framework, the components that are shown with a dashed outline are provided by the user, and the gray boxes show the logical grouping of the components in our framework into entities that may be on different hosts. The logical components of our framework are observers that perform and report observations, actors that perform actions, managers that receive observations, make decisions, and request actions, and a directory service for locating observers and actors. The next subsections describe these components in more detail.

2.1.1. Observer

An observer is a process on a computer system that provides information that can be measured from the system it is executing on. This could be information about the computer system, services or applications running on that computer system, or information that is not related to the computer system but that is accessible from that computer system. Examples of this last type of information are scheduling queue information from a front-end system and the current use of a local area network. An observer provides information in the form of *events*. An event has a type and contains data in the form of <name, value> pairs. An observer allows a manager to query for a single event or to subscribe for a set of events. A subscription is useful, for example, if a user wants to be notified of the load on a system every minute.

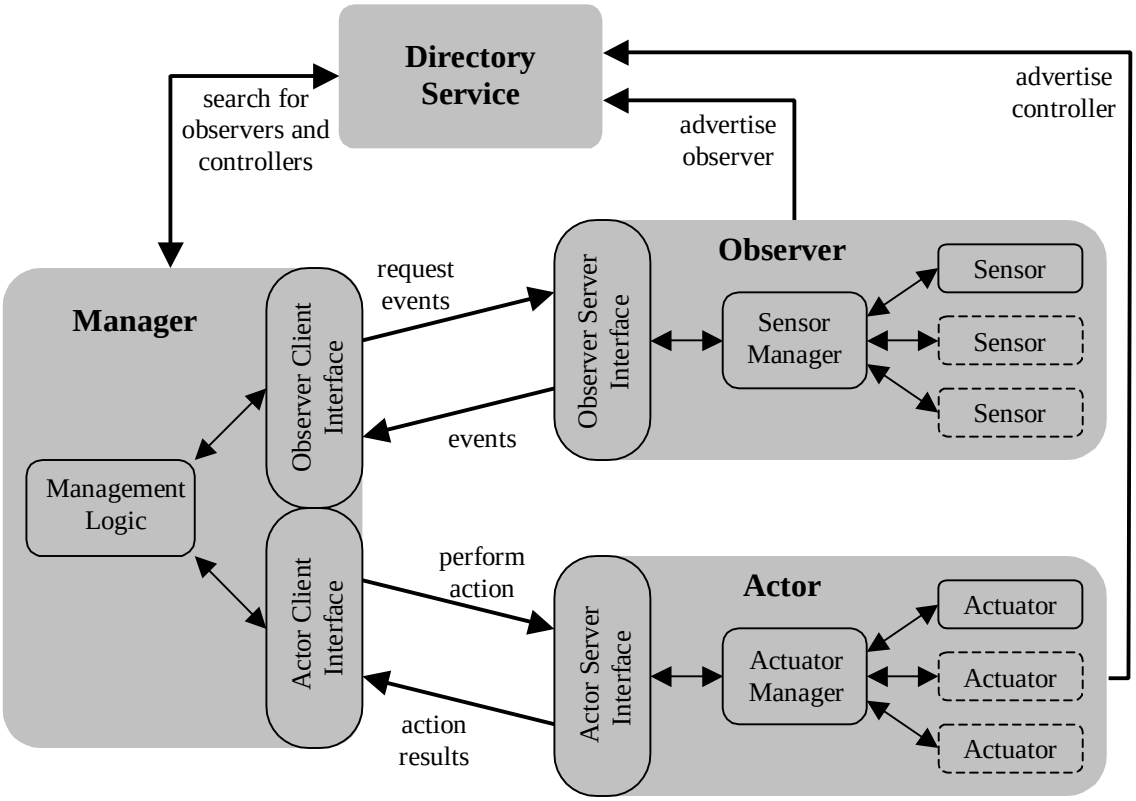

Figure 1. Architecture of the CODE framework.

An observer consists of the following components:

- **Sensor**. A sensor is a component that is used to sense or measure some property. For example, a CPU load sensor would measure the CPU load of a host, a network bandwidth sensor would measure the available bandwidth between two hosts, or a convergence sensor might measure the convergence rate of an application. A sensor is a passive component that does not perform any measurements until it is asked to by the sensor manager. We are providing a set of sensors as part of our framework, but users will most likely need to implement sensors for their specific purposes.
- **Sensor Manager**. The sensor manager receives event requests or subscriptions from the observer server interface, uses the appropriate sensor at the appropriate time to perform a measurement, and sends the result of the measurement to the observer server interface in the form of an event.
- **Observer Server Interface**. The observer server interface provides an interface for observers to access a distributed event service. This event service allows managers to subscribe or query events from observers and for observers to send events to managers.

2.1.2. Actor

An actor is a process on a computer system that can be asked to perform actions. These actions are made from the actor process and could affect local or remote resources, services, and applications. An actor consists of the following components:

- **Actuator**. An actuator is a component that can be used to perform a specific action. For example, an actuator can be used to start a daemon, submit a job to a scheduler, or change a variable in an application An actuator is a passive component that does not perform any actions until it is asked to by the actuator manager. We are providing a set of actuators as part of our framework, but users will most likely need to implement actuators for their own specific purposes.
- **Actuator Manager**. The actuator manager receives requests to perform actions from the actor server interface, uses the appropriate actuator to perform the action, and sends the results of the action back to the actor server interface.
- **Actor Server Interface**. The actor server interface provides an interface to a distributed action service that transmits requests for actions and their results. The purpose of the distributed action service is to allow a manager to request that an actor perform an action, and then transmits the results of the action back to the manager.

2.1.3. Manager

A manager is a process that asks observers for information, reasons upon that information, and asks actors to take actions when the observations indicate that actions need to be taken. A manager consists of the following components:

- **Management Logic**. The management logic receives events from the observer client interface, reasons upon this information to determine if any actions need to be taken, and then takes any actions using the actor client interface. There are two ways to implement the management logic:
    - Write C or C++ code that contains a series of if and case statements, a state machine, or whatever code is needed to perform the management functions.
    - Use an expert system and write management rules. We are experimenting with using an expert system to simplify the writing of managers. Without an expert system, the user must write a (potentially large) series of conditional statements to examine events, determine what they mean, and perform the appropriate actions. With an expert system, a user defines a (hopefully smaller) set of rules and the expert system uses these rules to reason on events and perform actions. We have incorporated the CLIPS expert system [1, 18] into our framework for this purpose. The management rules are written by the user and tell the expert system how to operate.
- **Observer Client Interface**. The observer client interface is used to request events from observers and receive those events.
- **Actor Client Interface**. The actor client interface is used to request that actors perform actions and to receive the results of those actions.

2.1.4. Directory Service

A common component of computational grids are directory services or grid information services [15, 27]. A directory service is searchable distributed database that is accessed using the Lightweight Directory Access Protocol (LDAP) [20, 21]. We use a directory service to store the

locations of observers and controllers, describe what types of observations or actions they provide, and allow managers to search for the observers and controllers that provide the information or actions they are interested in.

2.2. Implementation

We have initially implemented the CODE framework in C++ to take advantage of the features provided by object-oriented languages when writing modular and extensible code. One of the main design goals of our code is modularity so that the code can easily be extended and modified. For example, the definition of Sensor and Actuator interfaces allows us to easily implement a variety of sensors and actuators while the sensor manager and actuator manager components simply understand the Sensor and Actuator interfaces but can manage any type of sensor or actuator. Other examples are the Transport and Encoder interfaces that provide interfaces for transmitting and encoding a set of event and action messages. This allows us to hide the implementation of various techniques for transmitting and encoding data. At this point, the CODE framework supports communication using TCP, UDP, and SSL. The SSL interface uses OpenSSL [11] and is compatible with the Globus security mechanisms.

The protocol we use to communicate between observers and managers is compatible with the event protocol [25, 26] that is being defined in the Grid Forum Performance Working Group [6]. This protocol encodes data using the eXtensible Markup Language (XML) [14] and CODE uses the expat XML parser [4] to decode messages. Further, the format of the data CODE places in the directory service is compatible with the LDAP schemas [24] being defined in the Grid Forum Performance Working Group.

As we mentioned previously, we are using the CLIPS expert system [18] in this project and we are initially targeting the Linux, Solaris, and IRIX operating systems. We expect to port our code to other flavors of Unix and to the Cygwin system that provides an Unix-like environment for Microsoft Windows. The framework is implemented in a multi-threaded manner using pthreads.

3. Example Uses of CODE

This section describes two applications that we are developing that use the CODE framework. The first application provides observation and control of the resources and services that are part of a Globus-based computational grid. The second application is an alternative implementation of an LDAP-based grid information service.

3.1. Grid Control System

As computational grids grow, it becomes very difficult to ensure the correct operation of the large number of resources and services that make up a grid and to configure the services that are available on a grid. We are developing a Grid Control System (GCS) to assist with these tasks in a Globus-based grid such as the NASA Information Power Grid. Figure 2 shows the architecture of this system.

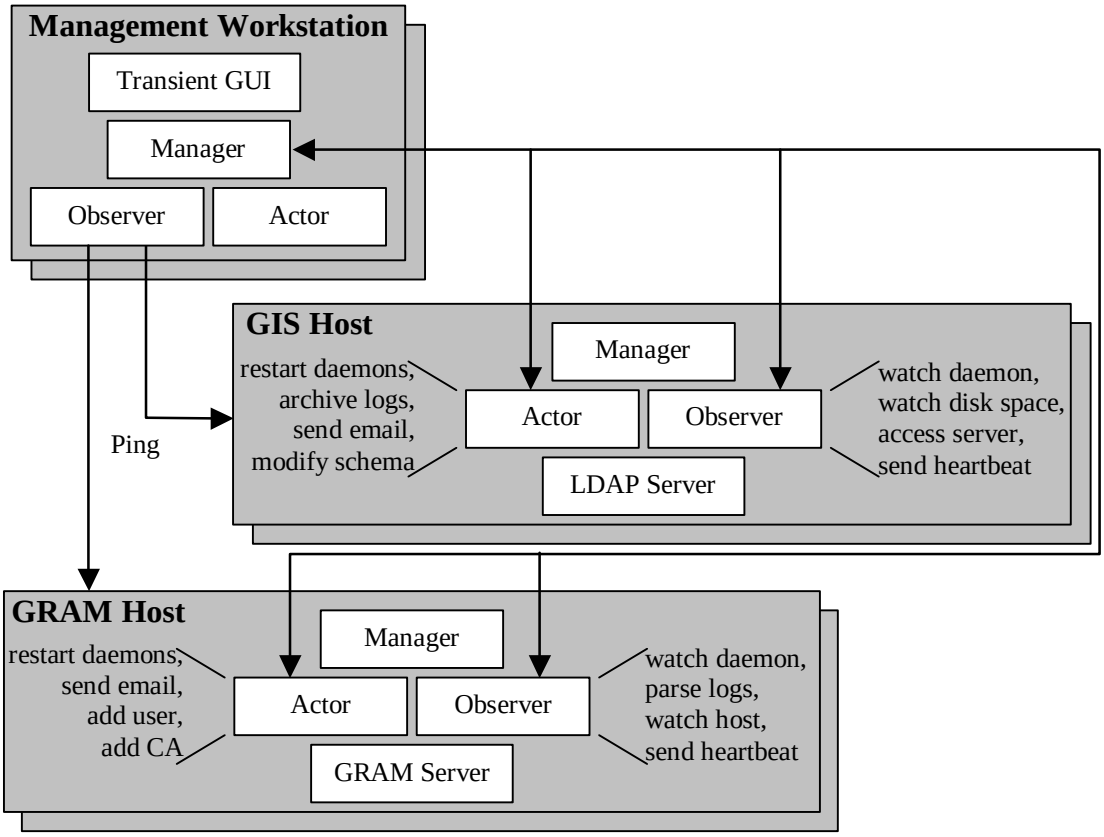

Figure 2. Architecture of the Grid Control System: A system to observe and control the resources and services of a Globus-based computational grid.

The figure shows CODE managers, actors, and observers on three different types of systems. First, a GRAM host is a computer system that is running the Globus Resource Allocation Manager (GRAM) service. This is the service that allows a user to execute an application on this computer system from a remote host. As you would expect, the GRAM service is widely deployed in Globus-based computational grids and this service, and the host it runs on, must be monitored and managed. The GRAM host is monitored so that an administrator knows that it is operating correctly: it hasn't crashed, it isn't overloaded, and so on. The GRAM service itself is monitored to determine if its daemons are executing, that it is updating the grid information service correctly, that users are not having problems executing applications, and so on. It is also very useful if the GRAM service can be configured from a remote location. This configuration includes adding or removing users, adding or removing certificate authorities, changing the certificate signing policy, and changing the amount of debugging information that is produced. In addition to these configuration actions, the actor on this system is used to start GRAM daemons, archive log files, and send emails.

To improve the robustness and scalability of the Grid Control System, a manager is running on each of the GRAM hosts. The purpose of this manager is to handle local problems, if it can, so that the relatively centralized management workstations do not have to be contacted to solve every problem. If the management workstations always have to be contacted, the robustness of the system would be reduced because the entire system would depend on always being able to

contact the management workstations and the scalability would be reduced because of the increased demands placed upon the management workstations. The manager on the GRAM hosts can be used to restart the local GRAM daemons if they crash, archive log files if they get too large, and so on.

The second system that has managers, actors, and observers is a GIS host. A GIS host is running an LDAP daemon that is part of an LDAP-based Grid Information Service (GIS). A GIS is used to store information about computer systems, networks, Globus daemons, installed software, applications, and so on. This information is necessary for the correct performance of many applications and services so it is critically important that it be available. The GCS can assist with the task of operating this service in several ways. The GCS can observe properties such as the load on the host, the status of the LDAP server processes, the size of the log files, the response time of the LDAP server, and the load being placed on the LDAP server (for some LDAP servers). This information allows a CODE manager or an administrator to notice potential problems and take corrective action. For example, if the LDAP server goes down, a CODE manager can restart it. If the log files are about to use up available disk space, they can be archived. If the response time of the LDAP server becomes unacceptably low, it can be reconfigured to limit the number of simultaneous users or to refer requests to other servers. Similarly to the configuration of the Grid Control System on a GRAM host, a GIS host also has a manager for performing local management tasks and improving the scalability of the system.

The third system that has managers, actors, and observers is a management workstation. A management workstation is used as a centralized point to gather information about the operation of a grid and to allow an administrator to control or configure a grid. The manager on this host can receive information from the observers on the GRAM and GIS hosts. In addition, the observer on a management workstation can be used to gather other data such as pinging GRAM and GIS hosts to see if they are operating. The actor on a management host can be used for actions such as sending emails when problems are seen.

A management workstation can be a bottleneck, but there are several steps that can help this situation. First, the GRAM and GIS hosts have managers to handle some of the problems that occur on those hosts. This reduces the communication traffic to a management workstation. Second, the rate at which information is sent to the management workstation by the observers on the GRAM and GIS hosts can be adjusted. As the number of GRAM and GIS hosts increases, the data rate from each host can be decreased. Third, a hierarchy of managers could be used with higher-level managers having less frequently updated information and/or summary information.

The management workstations also have a transient Graphical User Interface (GUI) that interfaces to the manager. The purpose of this GUI is to organize and present observations and actions to the administrator in a convenient manner. For example, instead of only displaying the list of users that have access to each system, this same data can be presented as the list of systems each user has access to. Similarly, instead of issuing one command to add a user to each system, a command to add a user to many systems at once can be provided. The GUI will also provide notifications to call attention to situations that need to be addressed using visual clues such as pop-up dialogs or flashing readouts or by sending email. These features improve the scalability of the system by increasing the number of resources and services that an administrator can manage.

3.2. An Alternative Grid Information Service

A computational grid needs an information service to store information about resources, services, users, applications, and so on. Currently, grid information services are accessed using the LDAP protocol and implemented using the open source OpenLDAP implementation or commercial implementations such as the one provided by IPlanet (formerly provided by Netscape). The LDAP protocol supports adding, deleting, and searching objects in a LDAP directory service. The data in an LDAP directory service is organized in a hierarchy called the directory information tree (DIT), usually based on real-world organizations and sub-organizations. For example, the root object may be for the Grid (o=Grid in LDAP terminology). This object would then have a NASA child object (o=National Aeronautics and Space Administration), which would have an Ames Research Center child object (ou=Ames Research Center), and so on. This organization means that every object in the database has an address that consists of the path to reach the object from the root of the tree.

When multiple servers are used to store the data in a DIT, this data is partitioned in a hierarchical manner. For example, the data for Ames Research Center (all objects under ou=Ames Research Center, o=National Aeronautics and Space Administration, o=Grid) might be located on a server at Ames. LDAP uses the notion of referrals to link LDAP servers. To continue the previous example, the LDAP server at Ames has a referral upward to the LDAP server for NASA. The NASA server also has referrals down to the Ames server, the Glenn server, and so on. This way a client making a query to the Ames server for information in the Glenn Research Center sub-tree would be referred to the NASA LDAP server that would then refer the client to the Glenn server.

One of the main disadvantages of LDAP server implementations is that the goal of improving search performance has resulted in slow and costly updates. Thus, if the data that is contained in an LDAP server is dynamic and must be updated frequently, the LDAP server may spend all of its resources updating its data and have no resources available to perform searches. This problem occurred in early Globus testbeds when there were only one or two LDAP servers used for all users of Globus. Two approaches were taken to address this problem.

First, the Globus group started developing the MDS-2 [13] grid information service. The second version of this implementation is about to be released, but both implementations are essentially hierarchies of OpenLDAP servers. The lowest-level servers in this hierarchy run on each host that has Globus GRAM services installed. These servers do not have traditional databases. Instead, when they are queried, they run a program to gather information from the host by mainly examining files created by Globus. The high-level servers receive search requests from users, forward these request to one or more child servers, merge the request results from the child servers, and pass the merged results back to the user. The high-level servers also cache the responses they receive from child servers for some period of time to improve performance. The advantages of this approach is that dynamic data is not moved from where it is generated until it is needed by a user. The main disadvantage of this approach is that users have found it to be unacceptably slow. This may result from a combination of the OpenLDAP implementation being relatively slow [27], and the fact that more servers are contacted because a user request typically goes from a high-level server, to a low-level server, to a high-level server, and then to the user.

Second, groups such as the IPG developers simply deployed multiple LDAP servers (typically commercial) so that the relatively frequent updates of information are not so concentrated on any

one server that they degrade search performance. The advantages of this approach are that high-performing commercial LDAP servers can be used. The disadvantages are that these servers can still be overwhelmed with updates and it can be the case that the dynamic information is rarely accessed by users, making the frequent updates irrelevant.

Our approach to providing a scalable LDAP-based grid information service is to extend the second approach by using a hierarchy of commercial LDAP servers to contain static information and older versions of dynamic information while using the CODE framework to obtain up-to-date dynamic information and to provide subscriptions. The goal of this implementation is to provide a higher-performing information service that combines the strengths of LDAP directory services and grid event services. The strengths of an LDAP directory service are that it can be distributed across many computer systems and it can be searched efficiently. The strengths of an event service, such as the one provided by CODE, are the low overhead to provide events and the ability to subscribe for events. The architecture of this system is shown in Figure 3.

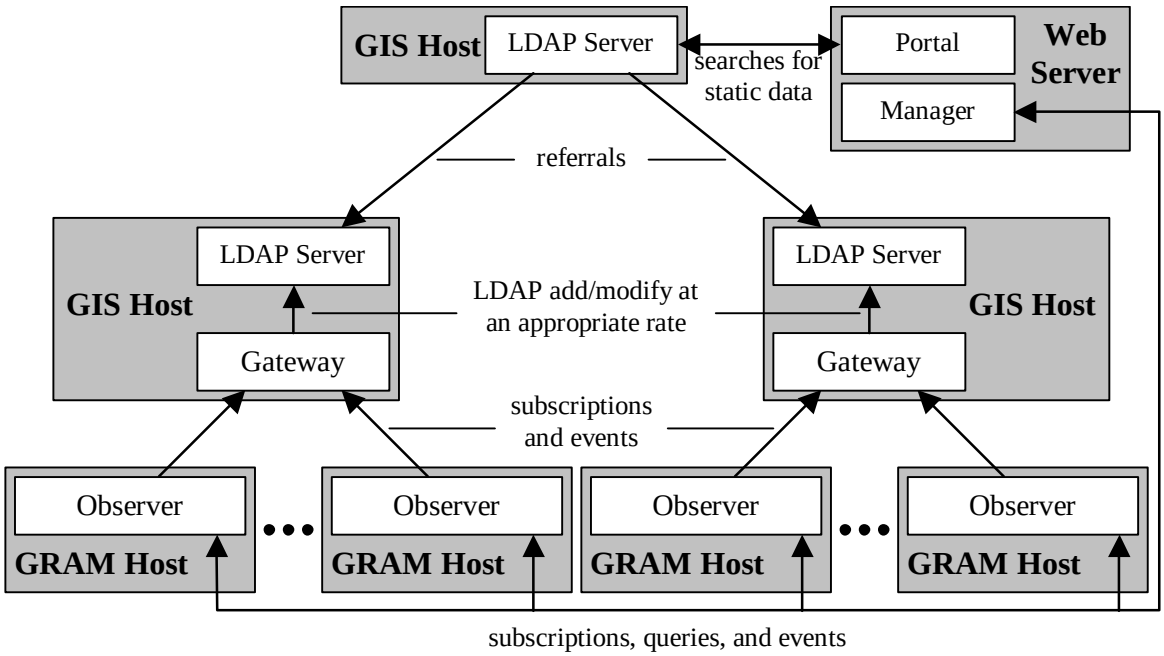

Figure 3. The architecture of an alternative grid information service that uses the CODE framework as the main information source.

This architecture shows that we are using a typical LDAP configuration with low-level LDAP servers containing data and higher-level servers referring to lower-level servers and perhaps containing data as well. We will use high-performing commercial LDAP servers in this implementation. To provide the information that is currently in grid information services, we will run a CODE observer on each computer system that is running a Globus GRAM server. These observers can provide information, in the form of events, about the computer system such as load, operating system, number of CPUs, scheduling information, information about the Globus deployment and any known Globus jobs, and so on. One of the entities that will receive this information is what we call a gateway. A gateway subscribes to observers for information and then writes this information into an LDAP server running on the same computer system. This

implies that the gateway is acting as a CODE manager and speaking the CODE event protocol to observers and is also an LDAP client and is modifying the data in the local LDAP server.

The gateway has two important functions. First, it translates each event it receives into an LDAP object at a specific location in the LDAP directory tree. Second, it manages the rate at which it modifies data in the LDAP server so that the LDAP server is not overwhelmed with updates. One way the gateway can accomplish this is by balancing the frequency each observer sends events with the number of observers that are sending events so that the overall update rate is less than some threshold. This threshold can be determined using experiments to measure the search performance obtained from an LDAP server under different update loads or the gateway can measure the search performance of the LDAP server as it executes. The gateway could also operate in a more sophisticated manner by using the access logs that are generated by many LDAP servers. These logs indicate which entries users accessed and therefore which entries the users are more interested in. The gateway can use this information to decide which entries (the ones more frequently accessed) should have more up-to-date information.

4. Summary and Future Work

Our efforts to deploy a computational grid at NASA have demonstrated the need for tools to observe and control the resources, services, and applications that make up grids. This has led to our development of CODE which provides a secure, scalable, and extensible framework for making observations from remote computer systems, transmitting this observational data to where it is needed, performing actions from remote computer systems, and analyzing observational data to determine what actions should be taken.

A prototype of our framework is complete and we are continuing to improve it. In addition to the core framework, we have implemented sensors for measuring various properties such as process status, file characteristics, disk space, CPU load, network interface characteristics, and LDAP search performance. We have also implemented a few simple actuators: “run a program” and “send an email”. We are currently developing two applications using CODE: a system to monitor and manage resources and services in large computational grids, and a grid information service. As part of these applications we are developing new sensors, new actuators, and application-specific components such as the event-to-LDAP gateways that are needed for the information service implementation.

In the future we will continue to improve and extend our framework as it is used for new applications. For example, we will add new sensors and actuators, we may incorporate new security mechanisms such as Kerberos, and we will most likely provide Java implementations of client libraries so that it will be easy to use this framework from GUIs. Further, we will track the standards for grid event services that are being developed in the Global Grid Forum and will strive to be compatible with those standards.

Acknowledgments

We gratefully acknowledge the help of Abdul Waheed who participated in the early phases of this project and of Jerry Yan who provided the initial motivation. We also wish to thank Dan Gunter, Ruth Aydt, Brian Tierney, Dennis Gannon, and Valerie Taylor for the many useful discussions we have had related to this work both inside and outside of the Grid Forum. This work is supported by the NASA HPCC/CAS program.

References

[1] "CLIPS: A Tool for Building Expert Systems," http://www.ghg.net/clips/CLIPS.html.

[2] "Condor High Throughput Computing," http://www.cs.wisc.edu/condor/.

[3] "The DOE Science Grid," http://www-itg.lbl.gov/Grid.

[4] "The Expat XML Parser," http://sourceforge.net/projects/expat/.

[5] "The Globus Project," http://www.globus.org.

[6] "Grid Forum Performance Working Group," http://www-didc.lbl.gov/GridPerf/.

[7] "The Legion Project," http://www.cs.virginia.edu/~legion/.

[8] "The NASA Information Power Grid," http://www.ipg.nasa.gov.

[9] "The National Computational Science Alliance," http://www.ncsa.uiuc.edu/access/index.alliance.html.

[10] "The National Partnership for Advanced Computing Infrastructure," http://www.npaci.edu/.

[11] "The OpenSSL Project," http://www.openssl.org.

[12] C. Catlett and L. Smarr, "Metacomputing," in *Communications of the ACM*, vol. 35, 1992, pp. 44-52.

[13] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing." In Proceedings of the The 10th IEEE International Symposium on High Performance Distributed Computing, 2001.

[14] D. Fallside, "XML Schema Part 0: Primer," http://www.w3.org/TR/xmlschema-0/.

[15] S. Fitzgerald, I. Foster, C. Kesselman, G. v. Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations." In Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing, 1997.

[16] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputing Applications*, vol. 11, pp. 115-128, 1997.

[17] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure,".: Morgan Kauffmann, 1999.

[18] J. Giarratano, "The CLIPS User's Guide,"., 1998.

[19] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. R. Jr., "Legion: The Next Logical Step Toward A Nationwide Virtual Computer," Department of Computer Science, University of Virginia CS-94-21, June, 1994 1994.

[20] T. Howes and M. Smith, *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*: Macmillan Technical Publishing, 1997.

[21] T. Howes, M. Smith, and G. Good, *Understanding and Deploying LDAP Directory Services*: MacMillan Technical Publishing, 1999.

[22] W. Johnston, D. Gannon, and B. Nitzberg, "Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid." In Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing, 1999.

[23] M. Litzkow and M. Livny, "Experience with the Condor Distributed Batch System." In Proceedings of the IEEE Workshop on Experimental Distributed Systems, 1990.

[24] W. Smith and D. Gunter, "Simple LDAP Schemas for Grid Monitoring," The Global Grid Forum GWD-Perf-13-1, 2001.

[25] W. Smith, D. Gunter, and D. Quesnel, "A Simple XML Producer-Consumer Protocol," The Global Grid Forum GWD-Perf-8-2, 2001.

[26] W. Smith, D. Gunter, and D. Quesnel, "An XML-Based Protocol for Distributed Event Services." In Proceedings of the The 2001 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, 2001.

[27] W. Smith, A. Waheed, D. Meyers, and J. Yan, "An Evaluation of Alternative Designs for a Grid Information Service." In Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing, 2000.